Directives

An OpenMP executable directive applies to the succeeding structured block. A structured-block is a block of executable statements with a single entry at the top and a single exit at the bottom, or an OpenMP construct.

Parallel [2.4]
The parallel construct forms a team of threads and starts parallel execution.

!$omp parallel [clause[ ] clause] ...
structed-block
!$omp end parallel
clause:
   if (scalar-logical-expression)
   num_threads: scalar-integer-expression)
default (private | firstprivate | shared | none)
private (list)
firstprivate (list)
shared (list)
copyin (list)
reduction (operator | intrinsic_procedure_name); list)

Loop [2.5.1]
The loop construct specifies that the iterations of loops will be distributed among and executed by the encountering team of threads.

!$omp do [clause[ ] clause] ...
do-loops
!$omp end do (nowait)
clause:
   private (list)
   firstprivate (list)
lastprivate (list)
reduction (operator | intrinsic_procedure_name); list)
schedule (kind, chunk_size)
collapse (n)
ordered
kind:
   static: Iterations are divided into chunks of size chunk_size. Chunks are assigned to threads in the team in round-robin fashion in order of thread number.
   dynamic: Each thread executes a chunk of iterations, then requests another chunk until no chunks remain to be distributed.
   guided: Each thread executes a chunk of iterations, then requests another chunk until no chunks remain to be assigned. The chunk sizes start large and shrink to the indicated chunk_size as chunks are scheduled.
   auto: The decision regarding scheduling is delegated to the compiler and/or runtime system.
   runtime: The schedule and chunk size are taken from the run-sched-var ICV.

Sections [2.5.2]
The sections construct contains a set of structured blocks that are to be distributed among and executed by the encountering team of threads.

!$omp sections [clause[ ] clause] ...
!$omp section
structured-block
!$omp end section
structured-block
...
!$omp end sections (nowait)
clause:
   private (list)
firstprivate (list)
lastprivate (list)
reduction (operator | intrinsic_procedure_name); list)

Single [2.5.3]
The single construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread).

!$omp single [clause[ ] clause] ...
structured-block
!$omp end single (end_clause[ ] end_clause) ...
clause:
   private (list)
   firstprivate (list)
end_clause:
copyprivate (list)
nowait

Workshare [2.5.4]
The workshare construct divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.

!$omp workshare structured-block
!$omp end workshare (nowait)
The structured block must consist of only the following: array or scalar assignments, FORALL or WHERE statements, FORALL, WHERE, atomic, critical, or parallel constructs.

Parallel Loop [2.6.1]
The parallel loop construct is a shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

!$omp parallel do [clause[ ] clause] ...
do-loop
!$omp end parallel do clause:
   Any accepted by the parallel or do directives with identical meanings and restrictions.

Parallel Sections [2.6.2]
The parallel sections construct is a shortcut for specifying a parallel construct containing one or more sections, and no other statements.

!$omp parallel sections [clause[ ] clause] ...
!$omp section
structured-block
!$omp end section
structured-block
...
!$omp end parallel sections
clause:
   Any of the clauses accepted by the parallel or sections directives, with identical meanings and restrictions.

Parallel Workshare [2.6.3]
The parallel workshare construct is a shortcut for specifying a parallel construct containing one workshare construct, and no other statements.

!$omp parallel workshare [clause[ ] clause] ...
structured-block
!$omp end parallel workshare
clause:
   Any of the clauses accepted by the parallel directive, with identical meanings and restrictions.

Task [2.7.1]
The task construct defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply.

!$omp task [clause[ ] clause] ...
structured-block
!$omp end task
clause:
   if (scalar-logical-expression)
   final (scalar-logical-expression)
   untied

Clause continues in next column.

Taskyield [2.7.2]
The taskyield construct specifies that the current task can be suspended in favor of execution of a different task.

!$omp taskyield

Master [2.8.1]
The master construct specifies a structured block that is executed by the master thread of the team.

!$omp master structured-block
!$omp end master

Critical [2.8.2]
The critical construct restricts execution of the associated structured block to a single thread at a time.

!$omp critical [name]
structured-block
!$omp end critical [name]

Barrier [2.8.3]
The barrier construct specifies an explicit barrier at the point at which the construct appears.

!$omp barrier

Taskwait [2.8.4]
The taskwait construct specifies a wait on the completion of child tasks of the current task.

!$omp taskwait

Atomic [2.8.5]
The atomic construct ensures that a specific storage location is updated atomically, rather than exposing it to the possibility of multiple, simultaneous writing threads. The atomic construct may take one of the following forms:

!$omp atomic read capture-stmt
!$omp end atomic
!$omp atomic write write-stmt
!$omp end atomic
!$omp atomic capture capture-stmt
!$omp end atomic
!$omp atomic update update-stmt
!$omp end atomic
!$omp atomic (update)
update-stmt
!$omp end atomic

capture-stmt, write-stmt, or update-stmt may be one of the following forms:

if is...
   read or capture = x
   write
   update, capture, or is = x = operator expr
   not present
   intrinsic_procedure_name = intrinsic_procedure_name(expr, expr_list)
   intrinsic_procedure_name is one of MAX, MIN, IAND, IOR, IEOR
   operator is one of *, /, -, 
   , OR, , AND, , EQV, , NEQV.

Flush [2.8.6]
The flush construct executes the OpenMP flush operation, which makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

!$omp flush (list)

(Directives continue >)
Directives (continued)

Ordered [2.8.7] The ordered construct specifies a structured block in a loop region that will be executed in the order of the loop iterations. This sequences and orders the code within an ordered region while allowing code outside the region to run in parallel.

!omp ordered
structured-block
!omp end ordered

Threadprivate [2.9.2] The threadprivate directive specifies that variables are replicated, each thread with its own copy.
!omp threadprivate(list)
Comma-separated list of named variables and named common blocks appearing between slashes.

runtime Routines

Execution Environment Routines [3.2]
The following execution environment routines affect and monitor threads, processors, and the parallel environment.

subroutine omp_set_num_threads( num_threads)
integer num_threads
Affects the number of threads used for subsequent parallel regions that do not specify a num_threads clause.

integer function omp_get_num_threads()
Returns the number of threads in the current team.

threadprivate clause.

integer function omp_get_max_threads() Returns the number of maximum number of nested active parallel regions. Valid values for dynamic are

threadprivate clause.

integer function omp_get_team_size() Returns the number of nested parallel regions enclosing the task that contains the call. Returns the value of max-active-variables-icv.

integer function omp_get_max_threads() Returns the value of max-active-variables-icv, which determines the maximum number of nested active parallel regions. Valid values for dynamic are

integer function omp_get_team_size() Returns the number of nested parallel regions enclosing the task that contains the call.

integer function omp_get_thread_num() Returns the value of the current thread. The thread number of the ancestor or the current thread.

integer function omp_get_thread_level() Returns, for a given nested level of the current thread, the thread number of the ancestor or the current thread.

integer function omp_get_thread_levels() Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.

integer function omp_get_thread_max_levels() Returns the number of nested, active parallel regions enclosing the task that contains the call.

logical function omp_in_final() Returns true if the routine is executed in a final or included task region; otherwise, it returns false.

Environment Variables

Environment variables are described in section [4] of the API specification. Environment variable names are upper case, and the values associated to them are case insensitive and may have leading and trailing white space.

OMP_DYNAMIC
Sets the dynamic adjustment of threads to use for parallel regions. Valid values for dynamic are true or false.

OMP_NUM_THREADS
Sets the number of threads for use for parallel regions. Valid values for dynamic are true or false.

OMP_PROC_BIND
Sets the value of the global bind-vars-icv. The value of this environment variable must be true or false.

Lock Routines [3.3]
The following lock routines support synchronization with OpenMP locks.

subroutine omp_init_lock(nvar)
integer(kind=omp_lock_kind) nvar
These routines initialize an OpenMP lock.

subroutine omp_destroy_lock(nvar)
integer(kind=omp_lock_kind) nvar
These routines ensure that the OpenMP lock is uninitialized.

subroutine omp_unset_lock(nvar)
integer(kind=omp_lock_kind) nvar
These routines provide a means of setting an OpenMP lock.

subroutine omp_set_lock(nvar)
integer(kind=omp_lock_kind) nvar
these routines set the lock but do not suspend execution of the task executing the routine.

subroutine omp_unset_lock(nvar)
integer(kind=omp_lock_kind) nvar
these routines provide a means of unsetting an OpenMP lock.

logical function omp_test_lock(nvar)
integer(kind=omp_lock_kind) nvar
These routines attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

Timing Routines [3.4]
The following timing routines support a portable wall clock timer.

double precision function omp_get_wtime()
Returns elapsed wall clock time in seconds.

double precision function omp_get_wtick()
Returns the precision of the timer used by omp_get_wtime.

Data Sharing Attribute Clauses [2.9.3]
Data-sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

default | firstprivate | shared | none
Default the default data-sharing attributes of variables that are referenced in a parallel or task construct.

shared(list)
declares one or more list items to be shared by tasks generated by a parallel or task construct.

private | firstprivate | none
private list
Declares one or more list items to be private to a task.

firstprivate(list)
declares one or more list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.

lastprivate(list)
declares one or more list items to be private to a task, and initializes each of them with the value that the corresponding original item had after the end of the region.

copyin(list)
Copies the value of the master thread's threadprivate variable to the threadprivate variable of each other member of the team executing the parallel region.

copyprivate(list)
Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

Data Copying Clauses [2.9.4]
Crcondition clauses are used to copy data values from private or threadprivate variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.

Operators for reduction (initialization values)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>.true.</td>
</tr>
<tr>
<td>-</td>
<td>.false.</td>
</tr>
<tr>
<td>and</td>
<td>.true.</td>
</tr>
<tr>
<td>or</td>
<td>.false.</td>
</tr>
<tr>
<td>max</td>
<td>Largest value</td>
</tr>
<tr>
<td>min</td>
<td>Smallest value</td>
</tr>
</tbody>
</table>

Data Copying Clauses [2.9.4]
These clauses support the copying of data values from private or threadprivate variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.

copyin(list)
Copies the value of the master thread's threadprivate variable to the threadprivate variable of each other member of the team executing the parallel region.

copyprivate(list)
Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

Environment Variables

Environmenv variables are described in section [4] of the API specification. Environment variable names are upper case, and the values associated to them are case insensitive and may have leading and trailing white space.

OMP_STACKSIZE size | K | M | G
Sets the stack-size-icv that specifies the size of the stack for threads created by the OpenMP implementation. size is a positive integer that specifies stack size. If unit is not specified, size is measured in kilobytes (K).

OMP_WAIT_POLICY policy
Sets the waits-policy-icv that controls the desired behavior of waiting threads. Valid values for policy are ACTIVE (waiting threads consume processor cycles while waiting) and PASSIVE.

OMP_MAX_ACTIVE_LEVELS size | K | M | G
Sets the max-active-levels-icv that controls the maximum number of nested active parallel regions.

OMP_THREAD_LIMIT size | K | M | G
Sets the thread-limit-icv that controls the maximum number of threads participating in the OpenMP program.